REPORT DOCUMENTATION PAGE			Form Approved OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	APRIL 2003	MARCH - APRIL 2003

4. TITLE AND SUBTITLE
GRAPHICS FILE CONVERSION: DADS "G.MOD" TO ".GEO" FORMAT

5. FUNDING NUMBERS

6. AUTHOR(S)
Wesley Bylsma

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

U.S. Army Tank-automotive and
Armaments Command/Research,
Development and Engineering Center
ATTN: AMSTA-TR-N/MS157
Warren, MI 48397-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

13863

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release: Distribution is unlimited.	A

13. ABSTRACT *(Maximum 200 Words)*

The following report documents the details of converting from the Dynamic Analysis and Design System (DADS) "g.mod" graphics file format to the ".geo" graphics file format. The file format of each type is discussed and the process described. A C program is included that automates the conversion process.

14. SUBJECT TERMS	15. NUMBER OF PAGES
graphics file conversion, g.mod, .geo, movie.byu	10
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unclassified

TR 13863

Graphic File Conversion: DADS “g.mod” to ”.geo” Format

Wesley Bylsma

U.S. Army Tank-automotive and Armaments Command
Research, Development and Engineering Center
ATTN: AMSTA-TR-N/MS157
Warren, MI 48397-5000

April 2003

Contents

List of Figures

List of Tables

Graphic File Conversion: DADS “g.mod” to ”.geo” Format

The following report documents the details of converting from the Dynamic Analysis and Design System (DADS) “g.mod” graphics file format to the “.geo” graphics file format. The file format of each type is discussed and the process described. A C program is included that automates the conversion process.

1 Introduction

Visualization of multibody dynamics is an important technique used to convey the physical interpretation of ground vehicle system performance. Many of the current multibody dynamics simulation codes interface with a graphics system to achieve this. Issues arise, however, when multiple simulation codes, each from a different vendor, are used for analysis. Each particular implementation handles storage of graphical entities differently. To allow the same graphical representation of a vehicle between different simulation codes requires graphic file format conversion. In this particular application, the Dynamic Analysis and Design System (DADS) multibody dynamics code uses an undocumented[1] “g.mod” format. This format needs to be converted directly to other formats for use in other programs for visualization. Conversion of this format directly to the “.geo” format is the topic of this report.

In order to better understand the process, we will begin with a description of the final format we desire. This will give us an objective that will help explain why we are doing what we are doing and provide a focus as we transform the “g.mod” format to the desired “.geo” format.

2 The “.geo” Format

The “.geo” format or “Movie.BYU” (“.byu”) file format originated from Brigham Young University. It describes a geometric shape in terms of its surfaces. It consists simply of an ASCII text file with fixed field sizes and positions.[2] When the number of values on a single line are exceeded, multiple lines are used. If the values on the last line of a section does not fill the line completely, that line is truncated. An example is shown in Figure 1.

The “.geo” file is composed of four main sections: header, parts, points, and polygons.

2.1 Header

The header section contains fixed size and position fields that give the overall extent of its contents. Table 1 gives the order and size for each part of the header section.

[1] LMS Intl. technical support claims that there is no written documentation to this format.
[2] see “lc.cray.com/doc/movie”.

Table 1: Header

No. of Parts	I8
No. of Points	I8
No. of Polygons	I8
No. of Lines	I8

The first field ("No. of Parts") gives the count for the total number of parts[3] in the file. The second field ("No. of Points") gives the count for the total number of points that make up the vertices of the line segments for each polygon face of each part.[4] The third field ("No. of Polygons") gives the count for the total number of polygons that make up all the parts. The fourth field ("No. of Lines") gives the count for the total number of point indexes that make up the line segments of each polygon face. Note that the same point may be referenced more than once, such as in adjoining edges of two polygons.

In the example given, Figure 1, there is one (1) part with eight (8) points (x,y,z) that make up 24 references to these points that consist of six (6) polygon faces. In C, the format for this section is

`"%8d%8d%8d%8d"`.

2.2 Parts

The Parts section contains the beginning and ending index of the polygons that make up that part as described in Table 2. The example in Figure 1 has only one part and thus contains only one entry[5]. One line can hold up to five (5) part entries.[6] For six (6) parts, the seventh entry would be on an additional line by

[3] This automatically implies the capability to contain multiple parts in one file and is pointed out here to make the reader aware–although it should be obvious.

[4] Points in this case refer to a set of x, y, and z values.

[5] One entries includes the beginning and ending part index so one entry actually consists of two values

[6] 5 * 2 numbers = 10 I8 numbers.

```
       1       8       6      24
       1       6
0.00000E+00-8.00000E+02-1.20000E+01 0.00000E+00-7.99000E+02 0.00000E+00
0.00000E+00 7.99000E+02 0.00000E+00 0.00000E+00 8.00000E+02-1.20000E+01
8.00000E+02-8.00000E+02-1.20000E+01 8.00000E+02-7.99000E+02 0.00000E+00
8.00000E+02 7.99000E+02 0.00000E+00 8.00000E+02 8.00000E+02-1.20000E+01
       5       6       7      -8       4       3       2      -1       5       1
       2      -6       6       2       3      -7       7       3       4      -8
       8       4       1      -5
```

Figure 1: Example ".geo File" File

Table 2: Parts

Beg. Polygon No.	I8
End.Polygon No.	I8

itself.[7] Notice that the part indexes assume a monotonic order. The number of entries in this section must match the number of parts defined in the header.

In C, the format of this section is

```
"%8d%8d"
```

per entry and can be repeated up to 5 times per line.

2.3 Points

The points section contains the two entries per line for each point.[8] The format

Table 3: Points

X Coordinate	E12.5
Y Coordinate	E12.5
Z Coordinate	E12.5

for each is given in Table 3. The number of entries in this section must match the number of points in the header. In C, the format of this section is

```
"%12.5E%12.5E%12.5E"
```

per entry and can be repeated twice per line.

2.4 Polygons

The polygon section contains a set of index values to points defined in the Points section the are the vertices of connected sets of points that compose line segments that make up the polygon faces of each part. A negative index value indicates the end of a sequence or polygon face definition. The beginning and ending points will automatically be connected to close the polygon face.

In the polygon section, up to ten (10) entries[9] per line are allowed. A full line would consist of 10 * I8 = 80 characters.

The number of entries in this section must match the number of lines in the header. The number of negative terminated sequences must match the number of polygons in the header. In C, the format for this section is

[7] Two entries = 2 * I8 = 16 characters

[8] Each point contains an x, y, and z value so two entries = 2 * 3 * E12.5 = 72 characters

[9] Each entry is defined in Table 4.

Table 4: Polygon

Point No.	I8

```
"%8d"
```

per entry and can be repeated up to 10 times per line.

3 The "g.mod" Format

While the "g.mod" format is undocumented, it is not too difficult to see where it contains the information needed to create a ".geo" file. Figure 2 shows an example of this format. (converted into the ".geo" format it is the same as Figure 1)

```
5C3663C5 (Magic_Number)
0 (Module_Node)
      8F (Relative_Seek_Value)
9 (Number_of_Nodes)
(Part_Name)
bump_pot
(Node_List)
0 0 0
0 -800 -12
0 -799 0
0 799 0
0 800 -12
800 -800 -12
800 -799 0
800 799 0
800 800 -12
1 (Module_Poly)
      75 (Relative_Seek_Value)
24 (Number_of_Edges)
(Part_Name)
bump_pot
(Polygon_List)
5 6 7 -8
4 3 2 -1
5 1 2 -6
6 2 3 -7
7 3 4 -8
8 4 1 -5
```

```
2 (Module_Pypnt)
      51 (Relative_Seek_Value)
6 (Number_of_Poly-pointers)
(Part_Name)
bump_pot
(Poly-pointers)
0 4 8 12 16 20
6 (Module_Plane)
      B3 (Relative_Seek_Value)
6 (Number_of_Plane_Equations)
(Part_Name)
bump_pot
(Plane_Equations)
1 0 0 -800 0
-1 0 0 0 0
0 -0.996546 0.0830455 -796.24 0
0 0 1 0 0
0 0.996546 0.0830455 -796.24 0
0 0 -1 -12 0
7 (Module_Geltype)
      51 (Relative_Seek_Value)
1 (Number_of_Geometry_Elements)
(Part_Name)
bump_pot
(Element_Type_List)
Extrude
```

Figure 2: Example "g.mod" File

Table 5 shows the matching section names for each format.

Table 5: Points

".geo"		"g.mod"
No. of Parts		Number_of_Geometry_Elements
No. of Points		Number_of_Nodes
No. of Polygons		Number_of_Poly-pointers
No. of Lines		Number_of_Edges
Points		Node_List
Lines		Polygon_List

Since the format is not specified, a "brute force" method is taken in reading the file. No optimization or performance claims are made. Each section is searched from the beginning of the file since no assumptions are made as to there relative location within it. Normal operation of the conversion program is to give an input and output file name. If the output file name (.geo) is not given, the program will replace the input file suffix (.mod) with (.geo) if no file of the same name already exists. It is assumed that the complete contents of the file can be read into memory since no memory paging is used. Data is read into memory with "read_gmod" and written out with "write_geo". White space is used as delimiters since no fixed format is defined.

3.1 Number_of_Geometry_Elements

As shown in Figure 2, this section corresponds to the "No. of Parts" section in Table 5. It gives the count for the total number of parts in the "g.mod" file. While this number is read in, it is assumed to be one (1)–only one part per file.

3.2 Number_of_Nodes

This section corresponds to the "No. of Points" section in Table 5. It gives the total number of points that make up the part.

3.3 Node_List

This section corresponds to the "Points" section in Table 5. It gives a list of the x, y, and z coordinates that makeup each point. There must be the same number in this list as indicated in section 3.2. Each point is on a separate line and the x, y, and z components are all on one line delimited by white space.

3.4 Number_of_Edges

This section corresponds to the "No. of Lines" section in Table 5 making up the edges composing the part. It gives the total number of index values to points that makeup the polygons.

3.5 Number_of_Poly-pointers

This section corresponds to the "No. of Polygons" section in Table 5. It gives the number of pointers into the "Polygon_List". Since only one part is assumed, the beginning polygon reference will be one (1) and the ending will be the same as this number. Under this assumption the "Poly-Pointer" section is not read in.

3.6 Polygon_List

This section corresponds to the "Lines" section in Table 5. It is an index into the "Node_List" for each point in the polygon. The end of a polygon is indicated by a negative value.

3.7 Plane_Equations

This section does not have a corresponding part in the ".geo" file format as outlined in Table 5. The ".geo" format uses the direction of the traversed points with the "right-hand-rule" to determine the normal direction. (in the face of the page, counter-clockwise would point up, clockwise down) Other formats, such as stereo lithography files handle normals directly–so we mention this section for interest. Five (5) coefficients per line correspond to the coefficients of the plane equation that define the normal. This section of the file is not used for this conversion process.

4 Summary

At this point we have established a relatively simple procedure to convert "g.mod" files into ".geo" files. The "g.mod" format differs slightly from the ".geo" format in that it does not have a fixed format, but is "free-form" (each line uses white space to delimit). It also contains normal information. Besides those two differences and different section name conventions embedded in the file–there is great similarity in content, which allows easy conversion.

References

[1] *DADS 9.6 User's Guide*, "2.2 What's in a Project?", LMS International (www.lmsintl.com).

[2] *http://myfileformats.com/* (movie.byu).

Appendix

A Program Listing

```
#include <stdio.h>
#include <stdlib.h> /* calloc,fgets */
#include <string.h> /* strstr */
/*---GMOD2GEO V1.0---
---remember .byu format not zero indexed but .wrl and C are!
---argv[0] = program name
   ---27FEB03 - created version 1.0
*/

#define LINE_LEN 1024

void read_gmod(FILE *in);
void write_geo(FILE *out);

typedef struct { float x,y,z; } P3;

/*---GEO GLOBALS---*/
int geo_nprt,geo_npts,geo_nply,geo_nlin;  /* parts, points, poly's, lines */
P3 *geo_pts;   /*geo_pts[npts];*/  /* parts */
int *geo_bprts, *geo_eprts;  /* beg and end of parts */
/*int *geo_ply;*/  /* poly's */
int *geo_lin;  /* lines (pts in poly) */
int geo_cntl;  /* counter for points and lines */

void main(int argc, char *argv[])
{
  FILE *in, *out;
  char margv2[LINE_LEN];
  char *ptr;
  /*int i; for (i = 0; i < argc; i++) printf("%d.[%s]\n",i,argv[i]);*/
  if (argc != 3)
  {
    strcpy(margv2,argv[1]);
    ptr = strstr(margv2,".mod");
    if (ptr != NULL)
    {
      strcpy(ptr,".geo");
    }
    else
    {
    printf("Use: program [input g.mod] [output.geo]\n");
    exit(1);
    }
}
  else
  {
    strcpy(margv2,argv[2]);
  }
  printf("\n\n\nGMOD2GEO 1.0\n\n");
  in = fopen(argv[1],"r");
  if ( (out = fopen(margv2,"r")) == NULL )
  {
    out = fopen(margv2,"w");
  }
  else
  {
    printf("File [%s] already exists.\n\n",margv2);
    exit(1);
  }

  printf("Processing %s ...\n",argv[1]);
  read_gmod(in);
  fclose(in);
  write_geo(out);
  fclose(out);

  free((P3 *)geo_pts);
  free((int *)geo_bprts);
  free((int *)geo_eprts);
  /* free((int *)geo_ply);*/
  free((int *)geo_lin);

  printf("\n\n\nDone.\n\n");
}

void read_gmod(FILE *in)
{
  char *valc=NULL;
  char line[LINE_LEN];
  int val = 0;
```

```
int i, j, k;
/*---READ GMOD DATAFILE AND SETUP PARAMETERS---*/
/*---GET PARTS---*/
/*---assume only one part---*/
rewind(in);
while (1)
{
  valc = fgets(line,LINE_LEN,in);
  if (valc <= 0)
  {
     break;
     printf("ERROR READING Number_of_Geometry_Elements\n");
 exit(1);
  }
  if ( strstr(line,"Number_of_Geometry_Elements") != NULL )
  {
    val = sscanf(&line[0],"%d", &geo_nprt);
    break;
  }
}

/*---GET NO NODES---*/
rewind(in);
while (1)
{
  valc = fgets(line,LINE_LEN,in);
  if (valc <= 0)
  {
     break;
     printf("ERROR READING Number_of_Nodes\n");
 exit(1);
  }
  if ( strstr(line,"Number_of_Nodes") != NULL )
  {
  val = sscanf(&line[0],"%d", &geo_npts);
    geo_npts=geo_npts-1;
    break;
  }
}

/*---GET NODE LIST---*/
rewind(in);
while (1)
{
  valc = fgets(line,LINE_LEN,in);
  if (valc <= 0)
  {
     break;
     printf("ERROR READING Node_List\n");
 exit(1);
  }
  if ( strstr(line,"Node_List") != NULL )
  {
    geo_pts = (P3 *) calloc(geo_npts,sizeof(P3));
    /*---allocate part poly index now---*/
    geo_bprts = (int *)calloc(geo_nprt,sizeof(int));
    geo_eprts = (int *)calloc(geo_nprt,sizeof(int));

    /*---READ IN PTS ---*/
    valc = fgets(line,LINE_LEN,in);
    for (i = 0; i < geo_npts; i++)
    {
      valc = fgets(line,LINE_LEN,in);
      val = sscanf(line,"%f %f %f", &(geo_pts+i)->x, &(geo_pts+i)->y, &(geo_pts+i)->z);
      if (val <= 0)
      {
       printf("ERROR READING Node_List\n");
  exit(1);
      }
    }
    break;
  }
}

/*---GET NO LINES---*/
rewind(in);
while (1)
{
  valc = fgets(line,LINE_LEN,in);
  if (valc <= 0) break;
  if ( strstr(line,"Number_of_Edges") != NULL )
  {
    val = sscanf(&line[0],"%d", &geo_nlin);
    if (val <= 0)
    {
     printf("ERROR READING Number_of_Edges\n");
exit(1);
    }
  }
}
```

```
  /*---GET NO POLY's---*/
  rewind(in);
  while (1)
  {
    valc = fgets(line,LINE_LEN,in);
    if (valc <= 0) break;
    if ( strstr(line,"Number_of_Poly-pointers") != NULL )
    {
      val = sscanf(&line[0],"%d", &geo_nply);
      break;
    }
  }

  /*---READ IN LINES AND CREATE POLY'S INDEX TO THEM---*/
  rewind(in);
  while (1)
  {
    valc = fgets(line,LINE_LEN,in);
    if (valc <= 0) break;
    if ( strstr(line,"Polygon_List") != NULL )
    {
      /*---READ IN LINS ---*/
      geo_lin = (int *)calloc(geo_nlin,sizeof(int));
      j=0;
      i=0;
      valc = fgets(line,LINE_LEN,in);
      if (valc <= 0)
      {
       printf("ERROR READING Polygon_List\n");
  exit(1);
      }
      while ( i < geo_nlin)
      {
        val = sscanf(&line[j],"%d%n", geo_lin+i,&k);
        if (val == -1)
        {
          valc = fgets(line,LINE_LEN,in);
          j=0;
          if (valc ==NULL)
          {
           printf("ERROR READING Polygon_List\n");
      exit(1);
          }
        }
        else
        {
          i = i + 1;
          j = j + k;
        }
}
      /*---assumes only 1 part---*/
      geo_bprts[0] = 1;
      geo_eprts[0] = geo_nply;
    }
  }
}

void write_geo(FILE *out)
{
  int i,geo_cntl;

  fprintf(out,"%8d%8d%8d%8d\n",geo_nprt, geo_npts, geo_nply, geo_nlin);

  /*---WRITE OUT PARTS DATA ---*/
  for (i = 0; i < geo_nprt; i++)
  {
    fprintf(out,"%8d%8d\n",*(geo_bprts+i),*(geo_eprts+i));
  }

  /*---WRITE OUT PTS ---*/
  for (i = 0; i < geo_npts; i=i+2)
  {
    if (i+1 < geo_npts)
    {
      fprintf(out,"%12.5E%12.5E%12.5E%12.5E%12.5E%12.5E\n", (geo_pts+i)->x, (geo_pts+i)->y, (geo_pts+i)->z,
                (geo_pts+i+1)->x, (geo_pts+i+1)->y, (geo_pts+i+1)->z);
    }
    else
    {
     fprintf(out,"%12.5E%12.5E%12.5E\n", (geo_pts+i)->x, (geo_pts+i)->y, (geo_pts+i)->z);
    }
  }

  /*---WRITE OUT LINES---*/
  geo_cntl = 0;
  for (i = 0; i < geo_nlin; i++)
  {
    fprintf(out,"%8d",geo_lin[i]);
    geo_cntl++;
    if (geo_cntl >= 10)
    {
```

```
      geo_cntl = 0;
      fprintf(out,"\n");
    }
  }
  fprintf(out,"\n");
}
```